Examining ASP.NET's Membership, Roles, and Profile - Part 1

By Scott Mitchell

Introduction

There's one thing messageboard websites, eCommerce websites, social network websites, and portal websites share in common: they all provide *user accounts*. These websites, and many others, allow (or require) visitors to create an account in order to utilize certain functionality. For example, a messageboard website, like ASPMessageboard.com, allows anonymous and authenticated visitors to view and search the posts in the various forums. However, in order to be able to post a new thread or reply to a message a visitor must have an account and must log into the site.

Providing user account support for a site involves the same set of steps: creating a database table to store user account information, creating a login page, defining a system by which authenticated users' logged on status is remembered across postbacks, specifying which pages are only available for authenticated users (authorization), creating a page for visitors to create a new user account, creating a page for the site's administrators to manage the user accounts, and so forth. Prior to ASP.NET, developers had to decide how to implement *all* of these facets on their own. ASP.NET introduced the concept of *forms-based authentication*, which provided a `FormsAuthentication` class to ease signing in and out of a site, as well as a protected *authentication ticket* to remember users' logged on status across page requests. (See Simple Authentication for an article on implementing authentication with classic ASP; refer to Using Forms Authentication in ASP.NET and Dissecting Forms Authentication for more information on ASP.NET's forms-based authentication capabilities.)

Even with forms-based authentication, though, ASP.NET developers are still on the hook for defining and creating the structure for storing user account information, for creating login and logout web pages, for enabling visitors to create new accounts and administrators to manage accounts, and so on. Thankfully ASP.NET version 2.0 has lightened developers' loads by providing the *membership* system and the security Web controls in ASP.NET 2.0. In a nutshell, membership is an API that provides programmatic access to common user account-related tasks. For example, there are methods to create a new user account, authenticate a user's credentials, delete a user, return all user information in the site, and so on. Furthermore, there are a number of security Web controls built atop this API that make performing common user account tasks as simple as dragging and dropping a control on the page.

In this article series we will be examining the ins and outs of version 2.0's membership, roles, and pofile systems and the various security Web controls. This particular article will examine the basics of membership with a look at configuring and using the built-in `SqlMembershipProvider`. As we will see, this particular provider stores user account information in a pre-defined database schema. Read on to learn more!

Forms-Based Authentication - A Step in the Right Direction, But Too Small a Step

Prior to ASP.NET, web developers were forced to define *all* of the authentication- and authorization-related decisions. One challenge was how to remember a user's logged on status across web requests; that is, after a user has successfully entered their username and password in the login, when they visit other pages how does the site remember that the user has already logged in? Another challenge was deciding how to protect pages from unauthorized access; that is, how can a page be configured to only allow a particular set of users, or only allow authenticated users? With classic ASP these challenges

were usually solved by using a Session variable to remember that a user had successfully logged in across web requests, and code on each page that checked this Session variable to determine the identity of the user visiting the page and whether or not they had access to visit the page.

To help reduce the work needed for implementing user accounts, ASP.NET version 1.0 included support for *forms-based authentication* as well as the ability to specify *authorization* rules in the `Web.config` file. The forms-based authentication provided a means to securely store an *authentication ticket* as a cookie in the user's browser to remember the user's logged on status across web requests. The `FormsAuthentication` class provides methods for working with this authentication ticket - both creating the ticket (logging on) and removing it (logging off).

Forms-based authentication aimed to provide a standardized approach for accomplishing two of the common tasks required for implementing user accounts. Unfortunately, forms-based authentication - on its own - still leaves the developer a lot of work. The developer must still decide how to serialize user account information, must still build the login page and write code using the `FormsAuthentication` class, must still create a logoff page, a page for creating a new account, a page for managing accounts, and so on. Not only did this require extra work on the developer's end, but also left important design decisions to the developer, decisions that might be made that did not follow best practices in design or security. (For example, a user might store users' passwords as plain-text in the database, rather than hashing or encrypting them. See this article for more information on the benefits of hashing passwords stored in a database.) The idea of ASP.NET version 1.0's forms-based authentication was great, but its implementation fell short.

Building On Top of Forms-Based Authentication with ASP.NET 2.0's Membership

Version 2.0 of ASP.NET takes the final step that forms-based authentication in version 1.x didn't - namely, it adds programmatic support for authenticating, adding, creating, deleting, and modifying user account information, along with Web controls for helping with accomplishing these tasks. Forms-based authentication is still alive and well in version 2.0, as well as the authorization settings in `Web.config`, and both are used the same way as in version 1.x. What 2.0 adds is the *membership* API and the security Web controls.

The membership API is implemented using the *provider model*, meaning that while the interface is well-defined, the actual implementation can be customized. The .NET Framework includes the `Membership` class that contains methods like `CreateUser()`, `GetAllUsers()`, `ValidateUser()`, and others. However, the actual class that is invoked when the API is used through an ASP.NET web application is based on the configuration of the application. You can provide your custom user account logic by creating a class that implements the defined membership API, and then configuring the web application to use your class. Of course you don't need to define a custom class - ASP.NET ships with two built-in membership providers, one that stores user account information in a SQL Server database, and another that uses Active Directory. Therefore, the membership system and security Web controls can be used with one of the built-in providers or, if you already have user data defined in some custom manner, a custom provider can be created so that the custom user store is utilized through the same API and security controls seamlessly. (See A Look at ASP.NET 2.0's Provider Model for more information on the concept behind the provider model; in a future article in this series we'll look at the steps involved in creating a custom membership provider.)

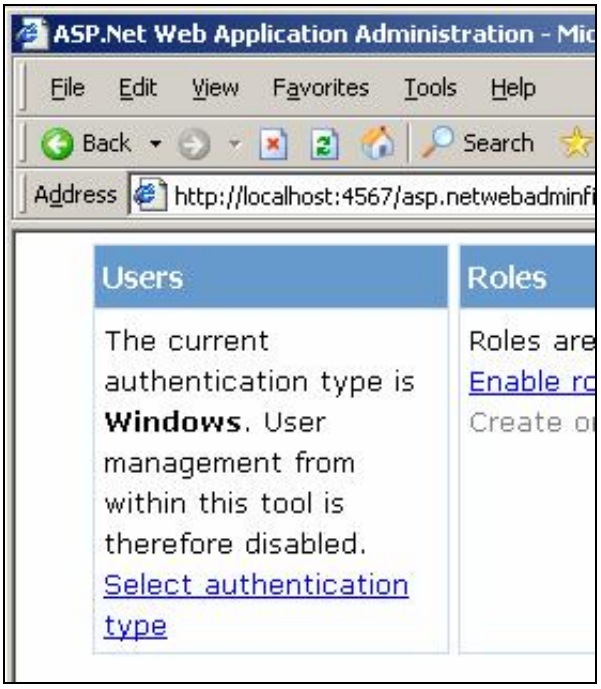
`SqlMembershipProvider` - Storing User Account Data in a SQL Server Database

One of the two membership providers that ship with ASP.NET 2.0 is the `SqlMembershipProvider` provider, which uses a SQL Server database to store authentication information. In order to use this provider you need to create the database schema used by the provider. There are two ways to accomplish this:

1. Use the ASP.NET Website Administration Tool (will create the database schema in a new SQL

Server 2005 database file `ASPNETDB.mdf`, placed in the application's `App_Data` folder)

2. Use the ASP.NET SQL Server Registration Tool (`aspnet_regsql.exe`) command-line tool (use this tool to implement the schema in a SQL Server 2000 or 2005 database)



To use the ASP.NET Website Administration Tool, start by launching the tool by going to Visual Studio's Website menu, choosing the ASP.NET Configuration menu option. Then, from the Security tab, change the authentication type to "From the internet," which can be accomplished either by clicking the "Select authentication type" link in the Authentication box or by clicking the "Use the security Setup Wizard to configure security step by step" link. Doing this will automatically create a database in your application's `App_Data` folder named `ASPNETDB.mdf` that has the predefined schema. (We'll examine this schema shortly.) You can also use the Website Administration Tool to specify authorization settings. For more on using the Website Administration Tool, see Website Administration Tool Overview, focusing on the Security tab.

If you want to store the user account information elsewhere - perhaps in a SQL Server 2000 database, or a SQL Server 2005 database not in the `App_Data` folder - you'll need to use the ASP.NET SQL Server Registration Tool (`aspnet_regsql.exe`) tool. This tool has a graphical component or can be used through the command-line. The graphical wizard allows you to specify the location to add the needed tables. For more information on using this tool refer to the technical documentation.

The Effects of the Administration Tool

When you use the Website Administration Tool to set the authentication type to "From the internet," it adds the following line to the site's `Web.config` file:

```
<authentication mode="Forms" />
```

If you create the schema through the ASP.NET SQL Server Registration Tool, you'll need to manually add this line to the `Web.config` file (or use the Website Administration Tool). Additionally, if you place the schema in a database other than `ASPNETDB.mdf` in the `App_Data` directory, you'll need to customize the membership configuration in the `Web.config`, specifying the connection string to the database.

The `SqlMembershipProvider` stores user account information in two related tables:

- `aspnet_Users` - has a record for each user account, storing the bare essentials. The `UserId` column uniquely identifies each user in the system, and is stored as a `uniqueidentifier` (a GUID).
- `aspnet_Membership` - has a `UserId` column that ties each record back to a particular record in `aspnet_Users`. The `aspnet_Membership` table stores core data associated with every user account: `Email`, `Password`, the security question and answer, and so on.

Customizing the `SqlMembershipProvider`

If you want to use the `SqlMembershipProvider` membership provider with the default settings (meaning that the user account information will be stored in the `ASPNETDB.mdf` SQL Server 2005 database in the `App_Data` folder), then you do not need to make any changes to `Web.config`, other than indicating that forms authentication should be used and specifying authorization rules. (These tasks can be done for

you by the Website Administration Tool, or manually. The syntax for ASP.NET version 2.0 is the same as it was in version 1.x. For more information on specifying authentication and authorization settings in `Web.config` refer to Authentication and Authorization and Authorizing Users and Roles.)

If, however, you want to use a different database, or want to change some of the membership settings (such as whether or not email addresses must be unique, the minimum password strength, if passwords are saved as plain-text, hashed, or encrypted, whether or not a security question and answer is required, and so on), you must specify the custom settings through a hand-entered block of XML in the `Web.config` file. (**Note:** you should *always* set a hard-coded value for the `applicationName` setting. See Scott Guthrie's Always set the "applicationName" property when configuring ASP.NET 2.0 Membership and other Providers blog entry for more information.)

The following chunk of XML shows how to customize the `SqlMembershipProvider` settings. Specifically, this XML in **bold** shows the `<membership>` element where the settings are customized. There's also a `<connectionString>` section that provides the connection string for the database that contains the schema. (Presumably this schema was added to this database using the ASP.NET SQL Registration Tool.)

```
<configuration>
  <connectionStrings>
     <add name="MyDB" connectionString="..." />
  </connectionStrings>
  <system.web>
    ... authentication & authorization settings ...

    <membership defaultProvider="CustomizedProvider">
      <providers>
         <add name="CustomizedProvider"
              type="System.Web.Security.SqlMembershipProvider"
              connectionStringName="MyDB"
              applicationName="ScottsProject"
              minRequiredPasswordLength="5"
              minRequiredNonalphanumericCharacters="0" />
      </providers>
    </membership>
  </system.web>
</configuration>
```

ASP.NET 2.0 includes a built-in connection string name `LocalSqlServer`, which points to the `ASPNETDB` database in the `App_Data` folder. If you want to keep using the default `ASPNETDB` database and only change a few properties, set the `connectionStringName` to `LocalSqlServer`.

In the `<membership>` a new provider is added named `CustomizedProvider` and made the default membership provider. This custom provider uses the `SqlMembershipProvider` still, it simply customizes some of the values, setting the `connectionStringName` to `MyDB` (as specified in the `<connectionStrings>` section), the `applicationName` to `ScottsProject`, the `minRequiredPasswordLength` to 5, and `minRequiredNonalphanumericCharacters` to 0. These are but a few of the customizable `SqlMembershipProvider` settings. For a complete list see `<add>` Element for Providers for Membership.

Always Set the `applicationName` Setting

You should *never* use the default Membership settings and, instead, always customize them to provide a static `applicationName` value. See Always set the "applicationName" property when configuring ASP.NET 2.0 Membership and other Providers for a discussion as to why.

Managing Users Through the Website Administration Tool

Once you have configured your ASP.NET website to use the membership system, you can manage your site's users through the ASP.NET Website Administration Tool. Simply go to the Security tab and click on the "Create User" link to create a new user account or "Manage Users" link to edit or delete existing user accounts. You can also active the roles feature from the online Administration Tool, as well as assign users to roles and roles to users. The role functionality is something we will examine in future articles in this series.

A Brief Overview of the Security Web Controls

ASP.NET 2.0 ships with all of the Web controls from version 1.x along with dozens of new ones. Of the plethora of new server controls, there are seven security Web controls. These seven controls provide a user interface for accomplishing user account-related tasks. Underneath the covers all of these controls use the membership system, meaning that even if you have a custom user store you can still use these controls if you create a custom membership provider (which will be the topic of a future article in this series!). The seven security user controls are:

- **Login** - presents the standard username/password login UI. By default, when the user clicks the "Login" Button a postback ensues and the control attempts to authenticate the user's supplied credentials by calling the `Membership` class's `VerifyUser(`*`username`*`, `*`password`*`)` method. If the credentials provided are valid, an authentication ticket is created for the user, otherwise an error message is displayed in the control's interface.

  To take custom steps when a user's credentials are invalid, create an event handler for the `LoginError` event; to implement your own authentication logic, create an event handler for the `Authenticate` event. The Login control contains a number of properties that can be configured to alter the appearance of the control's user interface. For complete control, use the `LayoutTemplate`. The following screenshot shows the Login control's default UI:

- **LoginView** - oftentimes on a page you want to display different content dependent on whether the page is being visited by an anonymous user or an authenticated user. For example, when an anonymous user visits the homepage you may want to show the Login Web control. However, when an authenticated user visits the same page you want to show a message like, "Welcome back, *username*," with a link to logoff.

  The LoginView control allows for such functionality, containing two templates, `AnonymousTemplate` and `LoggedInTemplate`. Simply place the Web controls and HTML markup you want anonymous users to see in the `AnonymousTemplate` and controls and markup for authenticated users in the

`LoggedInTemplate`. The LoginView also provides support for displaying different output based on the logged in user's role.

- **PasswordRecovery** - allows a user to receive their existing password or a new password sent to them through their email address on file. If the membership provider stores the password in a hashed format, then "recovering" the password actually creates a new, random password, which is then sent to the user. For passwords store as plain-text or encrypted, the actual, existing password is emailed to the user.
- **LoginStatus** - this simple control displays a link to the Login page if an anonymous user is visiting the page. If an authenticated user is visiting the page, a Logoff link is shown instead.
- **LoginName** - this control simply displays the username of the currently logged on user. The currently logged on user's name can also be accessed programmatically using `User.Identity.Name`, just as with ASP.NET version 1.x.
- **CreateUserWizard** - along with a page for logging in, every user account-enabled website also needs a page from which visitors can create a new user account. The CreateUserWizard control provides a UI for creating new user accounts. Similar to the Login control, after the user fills in the required fields and clicks the "Create User" button the `Membership` class's `CreateUser(...)` method is invoked. The CreateUserWizard can be customized with templates, as needed. The following screenshot shows the CreateUserWizard control in action:

The CreateUserWizard control's layout and inputs can be customized; see Customizing the CreateUserWizard Control for more information.

- **ChangePassword** - this control allows a user to change her password.

All of the security Web controls can be used without writing a lick of code. For example, to create a login page simply create a page named `Login.aspx` and drop a Login control on the page. Voila, you have a login page without writing a single line of markup or code. If you need to customize the appearance or logic used by these controls, you have complete control. The security controls can optionally use *templates*, meaning that you can specify the exact markup used to render the UI. Furthermore, the security controls expose a rich event model, allowing you to tap into the programmatic processing at

various points as needed. The download available at the end of this article contains pages showing how to use a number of these security Web controls; we'll examine these controls in much finer detail in future articles in this series.

Programmatically Using the Membership System

The membership system exposes its functionality programmatically through the `Membership` class, which has methods like `GetAllUsers()`, `CreateUser()`, `DeleteUser()`, and so on. This is the class used by the security Web controls to implement their functionality. You can use this class directly from your ASP.NET pages as well. For example, you can create a page that lists all of the users in the system by binding the results of the `Membership.GetAllUsers()` method to a GridView. If the built-in security controls aren't cutting the mustard, you can implement your own user interface and logic using the `Membership` class. See the download at the end of this article for some examples of using this class programmatically.

Conclusion

Since many websites provide user account support, it makes sense that support for such functionality be included in ASP.NET. With ASP.NET version 1.0, the goal was only half-attained. While forms-based authentication provided a standardized means for maintaining an authentication ticket to remember a user's logged on status across web requests, it did not provide any help in storing user account information or creating the necessary web pages (login, create an account, and so on). With version 2.0, ASP.NET now provides a membership service and security Web controls that complete what was lacking in version 1.x.

In this article we looked at the goal of the membership system and one of the built-in membership providers, `SqlMembershipProvider`. `SqlMembershipProvider` stores user account information in a SQL Server database, and can be customized through the web application's `Web.config` file. Furthermore, we overviewed ASP.NET's security Web controls, which make implementing login pages, create user account pages, and other user account-related user interface elements a walk in the park. Since the security Web controls interface with the underlying membership API, and since that API is customizable thanks to being built using the provider model, the security Web controls can be used against the `SqlMembershipProvider` or your own custom implementation.

The membership system is but one part of the overall user account picture in ASP.NET 2.0. There are also roles and profile systems that allow for role-based authorization and customized user account properties without the need for writing any code or creating any additional database tables. We'll examine membership and the security Web controls in further detail, along with the roles and profile systems, in future articles in this series.

Happy Programming!

- By Scott Mitchell

Attachments

- Download the code used in this article

Article Information	
Article Title:	ASP.NET.Examining ASP.NET's Membership, Roles, and Profile - Part 1
Article Author:	Scott Mitchell
Published Date:	December 7, 2005

Article URL:	http://www.4GuysFromRolla.com/articles/120705-1.aspx